Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2013, Article ID 428078, 10 pages
http://dx.doi.org/10.1155/2013/428078

Research Article

Performance Modeling for FPGAs: Extending the Roofline Model with High-Level Synthesis Tools

Bruno da Silva,[1] An Braeken,[1] Erik H. D'Hollander,[2] and Abdellah Touhafi[1,3]

[1] *INDI Department, Vrije Universiteit Brussel, 1050 Brussels, Belgium*
[2] *ELIS Department, Ghent University, 9000 Ghent, Belgium*
[3] *ETRO Department, Vrije Universiteit Brussel, 1050 Brussels, Belgium*

Correspondence should be addressed to Bruno da Silva; bruno.da.silva@vub.ac.be

Received 30 September 2013; Accepted 27 November 2013

Academic Editor: João Cardoso

The potential of FPGAs as accelerators for high-performance computing applications is very large, but many factors are involved in their performance. The design for FPGAs and the selection of the proper optimizations when mapping computations to FPGAs lead to prohibitively long developing time. Alternatives are the high-level synthesis (HLS) tools, which promise a fast design space exploration due to design at high-level or analytical performance models which provide realistic performance expectations, potential impediments to performance, and optimization guidelines. In this paper we propose the combination of both, in order to construct a performance model for FPGAs which is able to visually condense all the helpful information for the designer. Our proposed model extends the roofline model, by considering the resource consumption and the parameters used in the HLS tools, to maximize the performance and the resource utilization within the area of the FPGA. The proposed model is applied to optimize the design exploration of a class of window-based image processing applications using two different HLS tools. The results show the accuracy of the model as well as its flexibility to be combined with any HLS tool.

1. Introduction

Field programmable gate arrays (FPGAs) are a programmable and massively parallel architecture offering great performance potential for computing intensive applications. For applications with insufficient performance on a multicore CPU, FPGAs are a promising solution. However, the design effort for FPGAs requires detailed knowledge of hardware and significant time consumption. A performance analysis is required in order to estimate the achievable level of performance for a particular application, even before starting the implementation. Moreover, these models identify potential bottlenecks, the most appropriate optimizations, and maximum peaks of performance. On the other hand, the new generation of high-level synthesis (HLS) tools promised to reduce the development time and to automate the compilation and synthesis flow for FPGAs. By designing at high level, using C/C++ or even OpenCL languages, the compilers are able to generate parallel implementations of loops, containing large number of operations with limited data dependencies. Also, much of the debugging and verification can be performed at a high level rather than at the RTL code level, offering a faster design space exploration (DSE).

The purpose of this work is to present an insight model, in a similar fashion to the roofline model proposed by William et al. [1], where the highest performance and the potential bottlenecks of a particular algorithm are easily recognized on an FPGA. Most of the current HLS tools provide detailed information about the performance and an estimation of the resource consumption of each implementation. On one hand, the optimizations available on the HLS tools allow us to obtain a complete range of performance results for one algorithm. On the other hand, knowing the resource consumption of each design and the available resources of the target FPGA allows us to estimate the replication level. Together, the extended model provides an implementation guideline about the impact of the optimizations and about performance estimations considering the available resources.

This paper is organized as follows. Section 2 presents related work. The roofline model is introduced in Section 3.

The extended performance model for FPGAs is described in Section 4. The elaboration of the proposed model using HLS tools is detailed in Section 5. The calculation of the computational intensity is briefly explained in Section 6. In Section 7 the performance model is applied to window-based image processing and in Section 8 the model is generalized for any HLS tool. Conclusions are drawn in Section 9.

2. Related Work

Several analytic models have been proposed recently for multicore processors. The roofline model [1], proposed in 2008, is a visual performance model that makes the identification of potential bottlenecks easier and provides a guideline to explore the architecture. It has been proved to be flexible enough to characterize not only multicore architectures but also innovative architectures ([2–4]). In the GPU community the model has been well accepted ([5–7]), due to the similarity of GPU architectures and multicore processors.

Nevertheless, as modeling FPGAs demand a considerable number of parameters, the model has been considered for FPGAs just in a few cases [8, 9]. On the other hand, FPGA performance models have been already proposed in the past ([10] and especially [11]); however, the HLS tools were not mature enough at that time to be included in the model. A methodology and a model are proposed in [12] and extended in [13], obtaining interesting results using Impulse C. In [14] a mathematical model for pipelined linear algebra applications on FPGAs is presented. Finally, [15] presents the basis of the proposed model. However, in this paper we intend to offer further details and applications of our model.

3. The Roofline Model

The purpose of the roofline model is to provide a performance estimation to programmers and architecture designers. It is constructed considering bound and bottleneck analysis to visually predict realistic performance estimations. The original model expresses the maximum performance of an application running on an architecture as a function of the peak computational performance (CP) of the architecture and the reachable peak I/O bandwidth (BW). The CP consists of the maximum number of floating-point operations, measured in GFLOPs, that the processor or the functional unit is able to achieve.

The computational intensity (CI), which relates the CP and the BW, reflects the complexity of the algorithm, being the number of operations executed per byte accessed from memory. This model identifies the CP or the I/O BW as limiting factors. Therefore, the maximal attainable performance is always below the roofline obtained from both parameters:

$$\text{Attainable Performance} = \min(\text{CP}, \text{CI} \times \text{BW}). \quad (1)$$

Figure 1 summarizes the basis of the model, depicting several performance roofs, which are defined by the hardware specification or obtained through benchmarking. As depicted, Algorithm 1 has a low CI which makes the I/O BW the limiting factor. It means that the algorithm does not

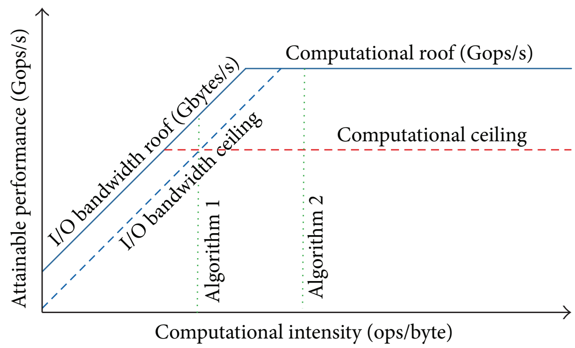


Figure 1: Basis of the roofline model. Considering two applications with a different CI, they are either compute or I/O bound.

perform enough operations per byte fetched from memory to hide all the memory latencies, and therefore the program is often spending more time waiting for data than generating new results. With a higher CI, Algorithm 2 is compute bound and does not consume all the available I/O BW. Furthermore, the model defines other boundaries, called "ceilings", that can only be overcome if the application exploits the available resources in an efficient way. An example of how the model is used as guideline to break through most of these ceilings is given in [16].

The roofline model is constructed from the hardware description of the multicore architecture. Unfortunately, the same approach cannot be directly applied for FPGAs because they are fully programmable technology, whereas the architecture of traditional processors is fixed. On FPGAs, the target algorithm defines the architecture, forcing to construct the model for each algorithm. However, the main principles of the roofline model can be adopted and extended, identifying the performance boundaries or guiding the use of the optimizations to achieve higher performance.

4. Extending the Roofline Model for FPGAs

The interesting feature of the roofline model is the fact that it is able to condense in one graph the peak computation, the peak memory bandwidth, and the computational intensity. Two parameters are based on the architecture and the last one is related to the application. The extended model for FPGAs demands a reorganization of all these parameters in order to keep the essence of the model. As mentioned before, due to the fact that it is the algorithm which defines the architecture, the peak computation must be related to the properties of the algorithm as well. Thus, while in the original roofline model the CP remains constant when the CI increases, the CP is now directly affected by the CI and varying as well. We now describe the behaviour of the most important concepts from the original roofline model in the context of FPGAs.

Operations. The performance units of the CP must be also oriented for FPGAs. The floating-point operations, proposed in the original model to show the computational power of microprocessors, have a prohibitive cost on FPGAs. These operations are often avoided by FPGA designers, adopting other numeric representations as fixed-point operations. For that reason, byte-operations [Bops], defining the number of operations per byte, is a good candidate. The byte-operations are general enough to cover different kinds of integer operations (as fixed-point) and detailed enough to represent the complexity of the algorithms in function of the number of operations per byte.

Scalability. The CP is now defined by the maximum attainable performance of the algorithm. On FPGAs, processing elements (PEs) contain all the required components to perform the functionality of the algorithm. One PE can be used to complete an entire computation, but more can be used to increase the performance. The replication of PEs in order to increase performance is what we call scalability (SC), which is defined by the resource consumption of each PE and by the available resources on the target FPGA as follows:

$$\mathrm{SC} = \left\lfloor \frac{\text{Available Resources}}{\text{Resource Consumption per PE}} \right\rfloor . \quad (2)$$

Therefore, the attainable performance per PE ($\mathrm{CP}_{\mathrm{PE}}$), together with the SC, defines the CP of a particular algorithm:

$$\mathrm{CP}_{\mathrm{FPGA}} = \mathrm{CP}_{\mathrm{PE}} \times \mathrm{SC}. \quad (3)$$

Applying (3) into (1) becomes

$$\text{Attainable Performance} = \min\left(\mathrm{CP}_{\mathrm{PE}} \times \mathrm{SC}, \mathrm{CI} \times \mathrm{BW}\right). \quad (4)$$

Thanks to the SC, the relation between the resource consumption and the computational performance is reflected in the model. Furthermore, a PE can be considered the basic component of the model, since it reflects the performance, the resource consumption, and even the CI, as is explained below.

I/O Bandwidths. The original roofline model just considers the off-chip memory as potential performance boundary. Consecutive reviews have increased the range of memory boundaries, by including other kinds of off-chip memory (e.g., L2 Cache, PCIe bandwidth, or network bandwidth). This wide range of memory BW, which will be considered as I/O BW, is usually available at the same time in FPGAs. The features of the algorithm define which I/O BW is the roofline and if I/O ceilings exist, depending on whether the communication is through a I/O protocol as PCIe, an off-chip memory, or both (e.g., GPUs). The extended model not only identifies the limiting interface but also estimates the I/O BW of the different ceilings. That is, if the communication with the FPGA is done through the network bus, the attainable performance is increased by using the PCIe bus.

Roofs and Ceilings. The main visual differences between the extended model and the original one are the rooflines and the ceilings. On one hand, the computational roofline would no longer be constant due to the resource consumption impact on the performance. On the other hand, multiple ceilings must be included, reflecting the impact of the large set of available optimizations for FPGAs. Additionally, FPGAs may incorporate multiple I/O interfaces, which are defined as multiple I/O ceilings in the model, dependent of the algorithm and its implementation.

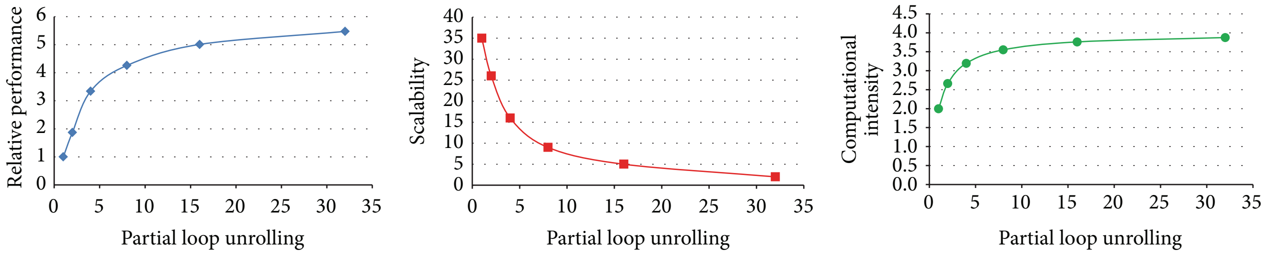
Computational Intensity. In the original model, the CI is usually increased through code adjustments or optimizations in order to avoid I/O BW performance bottlenecks. Adapting the model for FPGAs makes the CI the key how to construct the model. Once the CI is modified, the SC and the $\mathrm{CP}_{\mathrm{PE}}$ of each PE change as well. Thus, the evolution of the CI defines the final shape of the CP. The CI is modified through the available optimizations on the FPGAs. However, as mentioned in the original model, a way to increment the CI is by increasing the locality of the data. The importance of the locality resides on the high cost of the communication. The original way to maximize the locality is by using cache memory to minimize the off-chip communication. On FPGAs, the blocks of memory (BRAMs) can be the equivalent of cache memory. BRAMs are an internal dedicated resource of the FPGAs, which offer high BW. Thus, by using internal memory for data reuse, the CI increases because more operations can be done per external memory access. A higher CI means higher available external memory BW (by shifting to the right in the roofline model). The loop unrolling optimization is an example of how the internal memory increases the CI. The reuse of the input data in some algorithms by unrolling loops reduces the external memory accesses and increments the CI. Therefore, the BRAMs must be considered as any other resource on FPGA and, thus, can be a limiting factor for the final performance.

5. Using the High-Level Synthesis Tools to Construct the Model

Again, as only the available logic resources and the I/O BW are known in advance on the FPGAs, the proposed model can only be elaborated for one specific algorithm. Even for different designs of the same algorithm the model changes because only the implementation of the algorithm defines the resource consumption and the attainable performance. Without modern tools, this analytical performance model would require a large amount of effort rewriting the HDL algorithm description. However, thanks to the HLS tools, this task can be done much faster and easier nowadays since most of the tools offer different kinds of optimizations. Furthermore, most of the HLS tools provide reports with estimations of the FPGA resource utilization, latency, and throughput of the resulting RTL module. Besides the benefit of this additional information, the designer still has to decide, based on target specifications, what optimizations to use. However, the most adequate choice is not always evident. Based on Table 1, Figure 2 depicts the impact of the partial loop unrolling over several parameters as CI, SC, and relative performance. It shows how challenging the selection between all the available optimizations can be. By increasing the CI applying loop unrolling, the increment on the resource

Table 1: Generation of the extended model for one FPGA based on the resource consumption, the computational performance of one PE, and the I/O limited performance obtained using ROCCC. The resultant roofline is obtained applying (2) and (4).

Resource Consumption	No unrolling	Unrolling ×2	Unrolling ×4	Unrolling ×8	Unrolling ×16	Unrolling ×32
Slice registers (301440)	3652	6145	11132	21109	40573	79979
Slice LUTs (150720)	3157	4281	6335	10814	20189	37634
LUT-FF pairs (37680)	1069	1435	2245	3805	7193	13068
BRAM/FIFO (416)	1	2	3	4	8	15
DSP48 (768)	18	24	36	60	108	204
Max. number of PEs (SC)	35	26	16	9	5	2
Computational intensity (CI)	1.9768	2.624	3.144	3.496	3.704	3.816
Performance per PE (CP_{PE}) [GBops/s]	0.636	1.191	2.132	2.711	3.192	3.482
Computational performance (CP_{FPGA}) [GBops/s]	22.24	30.96	34.08	24.4	15.92	6.96
CI × PCIe × 8 BW [GBops/s]	8.302	11.027	13.190	14.678	15.554	16.0272
Resultant roofline	**8.302**	**11.027**	**13.190**	**14.678**	**15.554**	**6.96**



(a) (b) (c)

Figure 2: Impact of loop unrolling over the CI, the SC, and the relative performance. What is the most beneficial loop unrolling level analysing the values from Table 1?

consumption reduces the number of fittable PEs as well. However, there is a direct relation between the resource consumption and the offered performance per PE. Therefore, the question is what is the right level of loop unrolling in order to obtain the highest performance. The proposed model offers not only a visual performance estimation but also a guideline to reach the maximum performance through the available optimizations.

The elaboration of the proposed model using HLS tools is done through the DSE of an algorithm. Most of the HLS tools offer valuable information (resource consumption, latency, throughput, etc.) required to obtain the extended performance model. Figures 3 and 4 show how to construct the proposed model for a hypothetical algorithm. As first step, the performance of one PE and the initial CI is obtained. Figure 3 depicts how the CI increases by using optimizations that reuse fetched data from memory. The PEs with higher CI offer better performance, but their resource demands increase as well. The inclusion of the resource consumption into the performance model is done in Figure 4, showing how to obtain the computational roofline. This is one way to obtain the roofline by consecutively incrementing the CI and annotate the performance (CP_{PE}) and resource consumption (SC) of the implementation. On one hand, the CP_{PE} is obtained from the latency of the PE and the maximum operational frequency. On the other hand, the SC is obtained by considering the available resources of the FPGA and the resource consumption of one PE for different CI. Therefore, by applying (3), the value of the maximum attainable performance of the algorithm is obtained for each CI point reachable by the algorithm. Finally, it is interesting to notice that the attainable performance of one PE increases with the CI till some point, where the resource consumption starts to be the limiting factor, reflected in the SC. Therefore, higher CI does not necessarily imply higher attainable performance.

6. Obtaining the Computational Intensity

The CI is an important parameter to construct the proposed model since it reflects the complexity of an algorithm. However, this ratio is not so evident to obtain for complex algorithms. An analysis of the algorithm is mandatory to obtain this value. For simple algorithms it can be easily obtained. Although most of the compilers have enough information about the algorithm that they are compiling, we prefer to get the CI from high level, without considering the inner compiler operations. An algorithm is usually composed of one or several for loops and the loop body. We assume that each iteration of the loop represents one execution of the loop body. Therefore, if there is no data reuse between iterations, the CI obtained for the loop body would be the same as the complete algorithm, since the ratio would remain the same independently of the iteration. Thus, the CI is expressed as follows:

$$\mathrm{CI} = \frac{\#\text{Operations}}{\#\text{Inputs} + \#\text{Outputs}}. \tag{5}$$

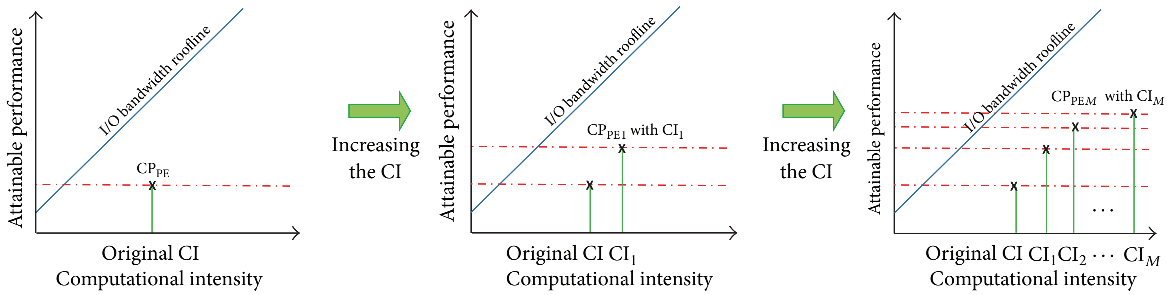


FIGURE 3: Impact of the increment of the CI in the extended model. By increasing the CI, thanks to the HLS optimizations, for example, the performance of one PE increases as well.

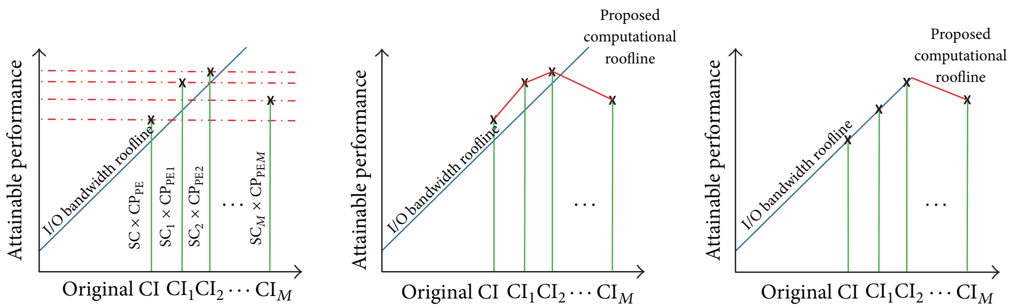


FIGURE 4: The benefit of a high CI is not so evident when the SC is also considered. The attainable performance is limited by the I/O bandwidth as well as the CP of the design.

Here #Operations are the number of operations involved in one iteration. As well, #Inputs and #Outputs are the required off-chip memory accesses for the input and output values, respectively. Once it is possible to reuse data between iterations, the CI can be increased. For the initial CI the iterations can be assumed independent. Then, the #Operations are obtained from the algorithm or even by monitoring the execution by placing additional counters to measure the total operations. Regarding the memory accesses, they are obtained using the same method as for #Operations. However, as soon as the CI starts to increase due to optimizations, the iterations can no longer be assumed to be independent. In that case, a study of the data reuse must be applied (see Case Study 1). However, some HLS tools provide enough information from where to extract the increased CI (see Case Study 2).

7. Case Studies

The proposed model is applied for two HLS tools: Riverside Optimizing Compiler for Configurable Computing (ROCCC) 0.6 [17] and Vivado HLS 2012.4. However, as our purpose is not the benchmarking of the tools, two different image processing algorithms with similar characteristics are used to elaborate the model with each tool. Both algorithms are implemented in a streaming fashion in a platform composed of two FPGAs Virtex6-LX240 on a Pico Computing backplane board EX500. The EX500 backplane board has a 16-lane PCIe Gen2 bus and accommodates 2 FPGAs, each with an 8-lane PCIe interface. By using two FPGAs in parallel, ×8 or ×16 PCIe lanes can be used, offering a wide range of I/O BW. The measured streaming bidirectional BW is 4.2 GB/s and 5.5 GB/s, respectively. Therefore, the available stream PCIe BW and the CP_{FPGA} roofline, obtained from the available resources, would be our rooflines. Finally, our results have been obtained processing a 1024×1024 greyscale image.

7.1. Case Study 1: ROCCC and Dilation. ROCCC 2.0 is an open source HLS tool for FPGA-based code acceleration which support as input code a subset of the ANSI C language. ROCCC tries to exploit parallelism of the C-based code minimizing clock cycle time by pipelining and minimizing area. The architectures generated by ROCCC consist of a number of modules that are connected via first-in first-out (FIFO) channels in a pipeline fashion. ROCCC differs from other HLS tools in that it uses the concept of data reuse [18]: when generating HDL code, it analyses access to arrays to find patterns that allow possible data reuse between loop iterations, in order to reduce the number of off-chip memory accesses. The generated hardware will contain the so-called smart buffers that exploit data reuse between loop iterations. This inherent optimization, which is always active, internally consists of registers that cache the portion of memory reused.

ROCCC is used to implement a morphological operation called dilation. This is a basic example of a computation that uses a moving window over a two-dimensional data structure. Given an input image and a rectangular mask (or kernel) that contains ones and zeros, the output image is determined by placing the kernel over the corresponding input pixel and determining the maximum value of the pixels which correspond to the positions of the kernel containing ones. Dilation may be categorized as a neighborhood to pixel algorithm. To simplify matters, we consider a square kernel of size 3 by 3, containing only ones, and the input images are in grayscale. Therefore, it is not necessary to read the kernel values. As eight comparisons must be done to generate each output pixel and nine input pixels need to be read, so the initial CI equals 8/10.

Due to the features of this algorithm, the smart buffers have a positive impact over the CI. To compute the pixel of the first column using a square mask of 3 by 3 elements, 9 pixels must fetch the *smart buffers*. For the rest of the columns, only 3 memory accesses are required thanks to the reusing of the prefetched pixels. This reuse of data increases significantly the CI of the dilation operation. Knowing how the smart buffers operate, it is possible to measure the increment. As the mask is full of ones, the CI is defined as follows:

$$\mathrm{CI} = \frac{\#\mathrm{ByteOperations}}{(\#\mathrm{Memory\ Accesses}/\#\mathrm{Bytes\ of\ the\ Image})}, \quad (6)$$

$$\begin{aligned} \mathrm{CI}_{\mathrm{ROCCC}} &= \frac{8}{\left(\left(H \times (k^2+1) + H \times (W-1) \times (k+1)\right) / (H \times W)\right)} \\ &= \frac{8 \times W}{(k^2+1) + (W-1)\cdot(k+1)}. \end{aligned} \quad (7)$$

Here H and W are the height and the width of the input image, respectively. Since we are assuming a square mask, k represents both dimensions of the dilation kernel, but the formula can be adjusted as well for nonsquared kernels. The first term of the denominator reflects the prefetching of the smart buffers while the other adder shows the additional fetches in the remaining steps. Notice as well that, for wide images, the *smart buffers* make the CI approach $8/(k+1)$. Thus, with $k = 3$ and without additional optimizations, the *smart buffers* increase the CI to about 2. For this particular case with $k = 3$ and as an example of their behavior, the smart buffers store 9 values and each iteration 3 new values are requested to the memory, since 6 remaining values can be reused. However, to process the first pixel of each row, the smart buffers need to be completely fetched with 9 new data.

In addition to the smart buffers, ROCCC offers other optimizations such as loop unrolling, which is able to increase the CI by reducing the memory accesses. Figure 5 shows an example of how unrolling the loop twice increases the reuse of data between the parallel iterations of the algorithm. Thus, extending (7), a generalized version considering the partial loop unrolling impact over the CI can be obtained:

$$\begin{aligned} \mathrm{CI}_{\mathrm{PLU}} = 8 \times N_{\mathrm{PLU}} \times W & \\ \times \left(\left(\left(N_{\mathrm{PLU}} + k - 1\right) \times k + N_{\mathrm{PLU}}\right) + (W-1)\right. & \\ \left. \times \left(\left(N_{\mathrm{PLU}} + k - 1\right) + N\right)\right)^{-1}. & \end{aligned} \quad (8)$$

Here N_{PLU} represents the level of loop unrolling. Figure 5 also shows how the memory accesses are reduced and the

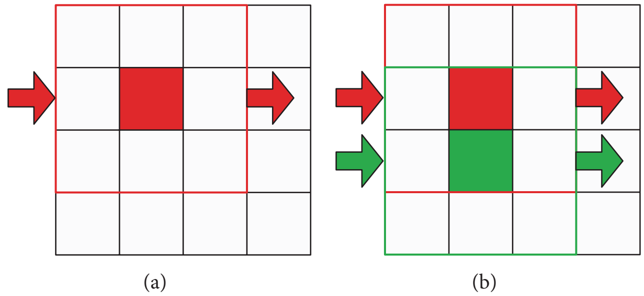


Figure 5: Example of how the partial loop unrolling reuse the fetched data. The kernel with $k = 3$ depicted in (a) requires 9 input values. By unrolling two times, consecutive rows are processed in parallel in (b). In that case, besides the fact that each iteration demands 9 input values, 6 values are shared between both iterations, reducing the memory accesses. As the smart buffers are active, only 4 new inputs, the last column of each window, need to be requested.

CI is increased. That is, by unrolling the loop 2 times the CI increases up to 2.56, and if the loop is unrolled further, the CI approaches 4.

Table 1 summarizes the elaboration of the proposed model. The VHDL code generated by ROCCC is synthesized using the Xilinx ISE 14.4 design software to obtain the resource consumption.

Once the resource consumption is obtained for each design, the most limiting resource is identified and the maximum number of PEs which can fit on the FPGA is estimated. The CI is obtained from the number of memory accesses and knowing that each output pixel requires 8 Bops. Finally, the reachable performance of each PE is obtained from the execution time operating in pipeline ($\mathrm{CP}_{\mathrm{PE}}$). Therefore, with both parameters, it is possible to derive the maximum attainable $\mathrm{CP}_{\mathrm{FPGA}}$. Finally, the minimum of the $\mathrm{CP}_{\mathrm{FPGA}}$ and the I/O BW limited performance for each CI defines the performance model.

Figure 6 shows the roofline obtained from the measurements. Instead of removing all the computational boundaries above the I/O bound, we prefer to keep it in order to clarify the proposed model. The dash lines are the ×8 PCIe BW and the computational roofline obtained for one FPGA, while the continuous lines are the ×16 PCIe BW and the computational roofline for two FPGAs. The vertical lines depicted on Figure 6 represent the obtained CI. The CI increases not only due to *smart buffers* but also with the loop unrolling optimization. This increment is beneficial since more I/O BW, which is the limiting factor, can be achieved. However, by further unrolling the loop, the latency of the operations and the resource consumption increase due to the internal memory consumption. In fact, after unrolling 16 times the resource consumption of each PE is so high that the attainable performance drops below the I/O limited performance.

7.2. Case Study 2: Vivado HLS and Erosion. Vivado HLS, former AutoESL's AutoPilot, is the HLS tool offered by Xilinx. Vivado HLS accepts C, C++, or SystemC as input and

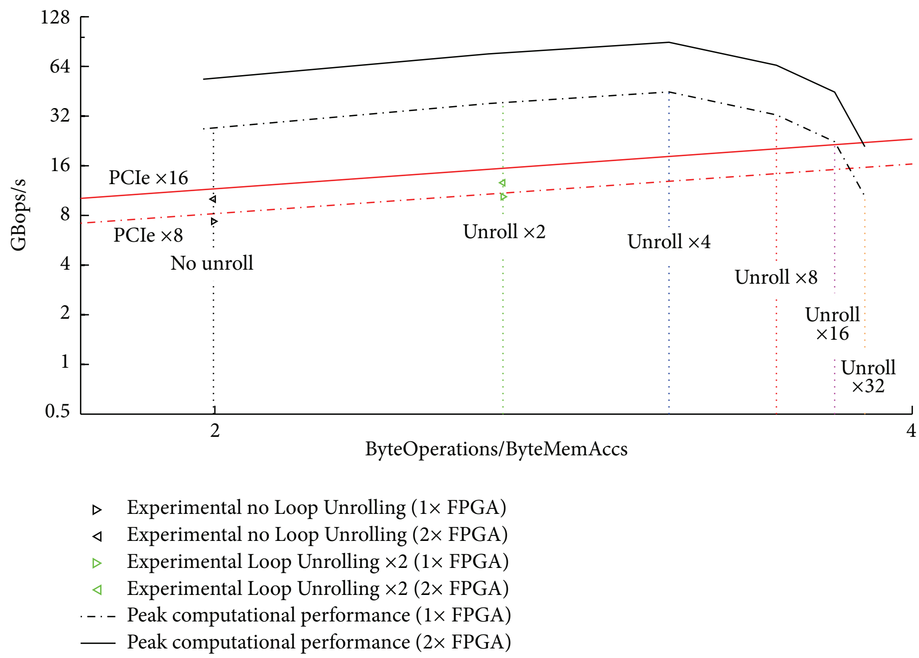


Figure 6: Superimposed performance models of one FPGA (dashed lines) and two FPGAs (continuous lines) of dilation using ROCCC.

converts each source code into a synthesizable RTL module. Vivado HLS tries to identify the constraints of the target Xilinx FPGA and generates optimized RTL using advanced platform-based code transformations and synthesis optimizations. Vivado HLS's extreme compatibility with C-based languages drastically reduces the number of modifications necessary to obtain synthesizable code. Besides the generated RTL code, Vivado HLS provides reports with estimations of the FPGA resource utilization, latency, and throughput of the resulting RTL module.

The algorithm implemented with Vivado HLS is another neighborhood to pixel algorithm called Erosion. In fact, it is the complementary morphological operation of dilation. As for dilation, one kernel of size 3 by 3, containing all ones and zeros, is applied to an input image. The resulting output pixel is the minimum value of the input pixels, which correspond to the positions of the mask containing ones. As explained before, the CI is again 8/10.

One of the main advantages of Vivado HLS for our model is its detailed report generated for each solution. The reports on Vivado HLS include not only one estimation of the resource consumption, but also accurate information about the latency. This information is used to get the SC and the CP_{PE}. Additionally, the analysis perspective offered by Vivado HLS makes the estimation of the CI of the algorithm possible by measuring the input and output off-chip memory accesses per iteration.

Besides, Vivado HLS does not include smart buffers as ROCCC; it accepts the use of line buffers, one kind of memory structure typically used to reduce external memory accesses on image processing algorithms. Our implementation of the model, however, just considers the optimizations available with the tool and no particular designs, using line buffers, which increase the CI.

In order to simplify the implementation of the first case study, we did not consider the impact of the frequency over the model. The model obtained with ROCCC achieves higher frequency than our operational frequency (250 MHz). Thus, in this case, the frequency has no direct impact on the performance. Therefore, as the basis of the model has been already introduced in the previous case study, the impact of the frequency is now introduced for Vivado HLS.

Figure 7 shows all the ceilings obtained by applying different optimizations and considering only one FPGA. Again, we prefer to keep the CP lines above the I/O bound in order to better understand the elaboration of the model. Firstly, it is possible to see the impact of the pipelining optimization. This optimization is able to reduce the latency of the algorithm, increasing the final performance per operation. As with ROCCC, the partial loop unrolling optimization is able to increase the CI by reducing the external memory accesses. Removing the impact of the *smart buffers* from (8), the evolution of the CI due to loop unrolling using Vivado HLS is as follows:

$$\mathrm{CI}_{\mathrm{PLU}} = \frac{8 \times N_{\mathrm{PLU}}}{(N_{\mathrm{PLU}} + k - 1) \times k + N_{\mathrm{PLU}}}. \qquad (9)$$

On the other hand, operating at higher frequency increases the performance, as expected. In fact, it is interesting, but not surprising, that higher performance is achieved by operating at higher frequency than by applying the pipeline optimization. For high values of CI, however, the increment of the resource consumption has a higher impact over designs operating at high frequency. The additional computational rooflines indicate that the I/O BW cannot be the limiting bound. In fact, that is the case by operating at low frequency or without loop unrolling combined with pipelining. However, it has an impact over the resource

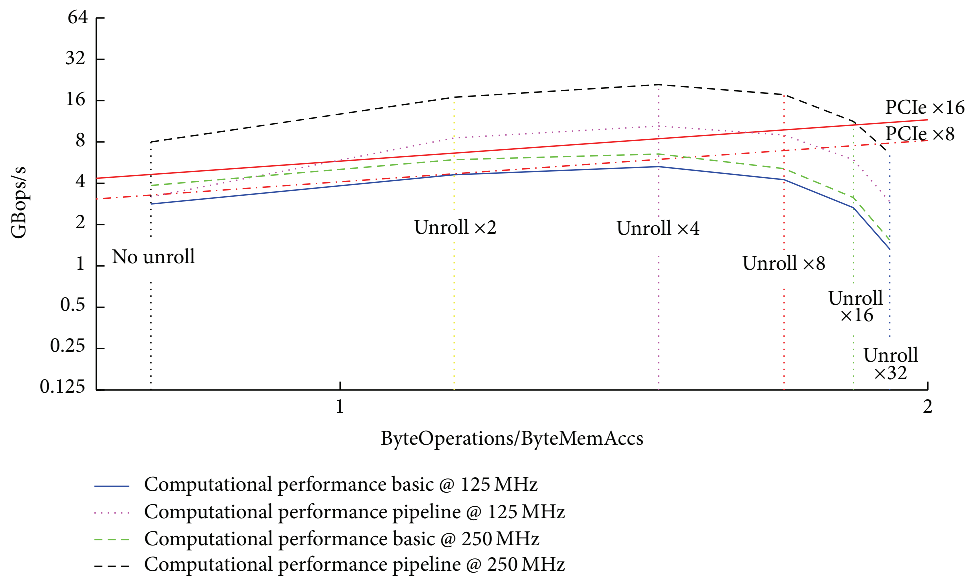


FIGURE 7: Roofline model including several designs of Erosion using Vivado HLS and with one FPGA.

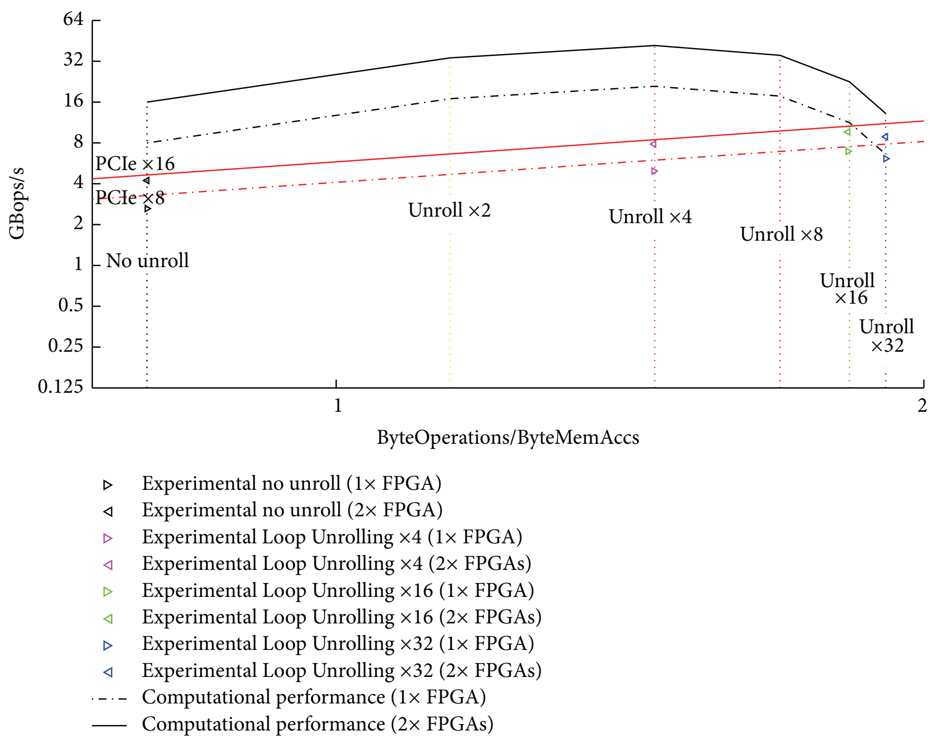


FIGURE 8: Resultant roofline model of Erosion using Vivado HLS and for multiple FPGAs. The experimental results depicted have been obtained with the pipelining design operating at 250 MHz.

consumption, especially for high values of CI, which must be considered.

Figure 8 depicts our experimental results by pipelining and operating at the highest frequency, since this combination offers the highest performance. Moreover, it also includes the results obtained for two FPGAs, where the dash lines are the ×8 PCIe BW and the CP obtained for one FPGA and the continuous lines are the ×16 PCIe BW and the CP for two FPGAs. Both figures show that, for higher CP, by unrolling 32 times, the performance drops below the available PCIe BW. The proposed model not only shows the relation between the resource consumption and the performance but also estimates the impact of the optimizations over a particular design, helping to identify realistic limits and attainable performances.

8. Generalizing the Model for Other HLS Tools

The main purpose of the case studies is to exemplify how the model is flexible enough to be elaborated for different HLS tools. Of course, the effort to elaborate the model is directly related to the features of the HLS tool. Thus, while Vivado HLS offers a complete report with most of the information required to construct the model, ROCCC offers interesting optimizations to reduce the memory accesses but does not provide any kind of report. However, the main concept of the model is not linked to any particular HLS tool and its elaboration can be adapted to any HLS tool, even when this does not provide any information further than the resultant HDL code. The acquisition of the parameters to construct the model, detailed in Section 5, can be summarized as follows.

(i) Resource estimation: these values can be obtained from any FPGA vendor tool as ISE/Vivado or Quartus II. For a fast elaboration, the synthesis report is enough, but for more accurate results, the placement and routing must be considered.

(ii) CI: this parameter can be obtained from the HLS tool, as in Vivado HLS and its analysis perspective, or it can be manually obtained as in the case of ROCCC. Besides, the effort to acquire this parameter is extremely dependent on the HLS tool; once the CI is obtained it can be extrapolated for algorithms with similar memory-access pattern.

(iii) Performance: again, the performance estimation can be obtained from simulation, as in the case of ROCCC, or from the report of the HLS tool, as for Vivado HLS.

The proposed model can even be elaborated without HLS tools, but the effort is too high without the fast DSE that the current HLS tools offer. The fact that some HLS tools do not provide enough information is not a main inconvenience for the model but for the user. This is who has to estimate the performance of a design without a HLS report, information that only other tools as ISE/Vivado or Quartus II can provide. Fortunately, the tendency of new releases of the HLS tools is to provide more complete and accurate information about resource consumption, latency, and frequency in their report. This fact drastically reduces the effort to elaborate the proposed model. However, thanks to this kind of tools, the model can be automatized and generated in a few minutes. In fact, this feature is considered as our future step.

Due to the strong link between the elaboration of the model and the HLS tool, the model provides not only a valuable performance estimation but also an estimation about what performance can be obtained for a particular algorithm and a HLS tool. Exactly the same high-level description of the algorithm can be translated in a high-performance design with a different HLS tool. For that reason, we believe that this analytical model is more linked to the FPGA or the HLS tools than to the algorithm, as the same way as an larger FPGA or a new release of an EDA tool provide better performance. Therefore, the proposed model is a good complement to the current HLS tools, providing an insight visual performance estimation and linking the performance estimation from the HLS tool with the available I/O BW.

Finally, the model can be reused for algorithms with similar memory-access pattern. A future improvement is a fast memory-access pattern recognition in order to reduce the range of optimizations to be tested. An early pattern recognition allows a significantly faster elaboration of the model since only the most promising optimizations are analysed.

9. Conclusions

The roofline model has proven to be flexible enough to be extended for FPGAs. By analysing the main characteristics of the original roofline model we missed the connection between the computational power and the resource consumption, which is one of the most important parameters on FPGAs. As a solution, the extended model combines the basis of the roofline model with the main characteristics of FPGAs. Also, the use of the HLS tools to elaborate the model reduced drastically the construction effort.

The model has been applied to a couple of window-based image processing algorithms using so different HLS tools as ROCCC and Vivado HLS. In both cases, the model provides a visual performance analysis considering the available resources, the attainable I/O BW, and the most interesting optimizations. The results also show that all the information collected from the report of the HLS tool or from other synthesis and EDA tools in order to elaborate the model is accurate enough to provide a realistic performance estimation. Our future steps are to automate the construction of the model, an easier extraction of the CI of the algorithm, and the exploration of the model for more complex algorithms.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[2] Y. Sato, R. Nagaoka, A. Musa et al., "Performance tuning and analysis of future vector processors based on the roofline model," in *Proceedings of the 10th Workshop on MEmory Performance: DEaling with Applications, Systems and Architecture (MEDEA '09)*, pp. 7–14, ACM, September 2009.

[3] M. Reichenbach, M. Schmidt, and D. Fey, "Analytical model for the optimization of self-organizing image processing systems utilizing cellular automata," in *Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW '11)*, pp. 162–171, Newport Beach, Calif, USA, March 2011.

[4] J. A. Lorenzo, J. C. Pichel, T. F. Pena, M. Suarez, and F. F. Rivera, “Study of Performance Issues on a SMP-NUMA System using the Roofline Model,” in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA ’11)*, Las Vegas, Nev, USA, 2011.

[5] H. Jia, Y. Zhang, G. Long, J. Xu, S. Yan, and Y. Li, “GPURoofline: amodel for guiding performance optimizations on GPUs,” in *Proceedings of the 18th International Conference on Parallel Processing (Euro-Par ’12)*, pp. 920–932, 2012.

[6] K. H. Kim, K. Kim, and Q. H. Park, “Performance analysis and optimization of three-dimensional FDTD on GPU using roofline model,” *Computer Physics Communications*, vol. 182, no. 6, pp. 1201–1207, 2011.

[7] C. Nugteren and H. Corporaal, “The boat hull model: adapting the roofline model to enable performance prediction for parallel computing,” in *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP’12)*, pp. 291–292, February 2012.

[8] M. Spierings and R. van de Voort, “Embedded platform selection based on the roofline model: applied to video content analysis,” 2012.

[9] B. da Silva, A. Braeken, E. H. D’Hollander, A. Touhafi, J. G. Cornelis, and J. Lemeire, “Performance and toolchain of a combined GPU/FPGA desktop,” in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA ’13)*, p. 274, 2013.

[10] J. Park, P. C. Diniz, and K. R. S. Shayee, “Performance and area modeling of complete FPGA designs in the presence of loop transformations,” *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1420–1435, 2004.

[11] L. Deng, K. Sobti, Y. Zhang, and C. Chakrabarti, “Accurate area, time and power models for FPGA-based implementations,” *Journal of Signal Processing Systems*, vol. 63, no. 1, pp. 39–50, 2011.

[12] B. Holland, K. Nagarajan, and A. D. George, “RAT: RC amenability test for rapid performance prediction,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 1, no. 4, article 22, 2009.

[13] J. Curreri, S. Koehler, A. D. George, B. Holland, and R. Garcia, “Performance analysis framework for high-level language applications in reconfigurable computing,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 3, no. 1, article 5, 2010.

[14] S. Skalicky, S. Lopez, M. Lukowiak, J. Letendre, and M. Ryan, “Performance modeling of pipelined linear algebra architectures on FPGAs,” in *Reconfigurable Computing: Architectures, Tools and Applications*, pp. 146–153, Springer, Berlin, Germany, 2013.

[15] B. da Silva, A. Braeken, E. H. D’Hollander, and A. Touhafi, “Performance and resource modeling for FPGAs using high-level synthesis tools,” PARCO, 2013.

[16] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick, “Optimization of a lattice Boltzmann computation on state-of-the-art multicore platforms,” *Journal of Parallel and Distributed Computing*, vol. 69, no. 9, pp. 762–777, 2009.

[17] J. Villarreal, A. Park, W. Najjar, and R. Halstead, “Designing modular hardware accelerators in C with ROCCC 2.0,” in *Proceedings of the 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM ’10)*, pp. 127–134, Charlotte, NC, USA, May 2010.

[18] Z. Guo, B. Buyukkurt, and W. A. Najjar, “Input data reuse in compiling window operations onto reconfigurable hardware,” in *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES ’04)*, pp. 249–256, June 2004.